ACEEE Int. J. on Electrical and Power Engineering, Vol. 4, No. 1, Feb 2013

Systematic Model based Testing with Coverage Analysis

Divyesh Divakar[1] and Ashwini K G[2]
[1, 2] Assistant Professor, Dept. of Electrical and Electronics Engineering
Canara Engineering College, Mangalore, India
divyesh_divakar@rediffmail.com, ashwini_shenoy@rediffmail.com

***Abstract*— Aviation safety has come a long way in over one hundred years of implementation. In aeronautics, commonly, requirements are Simulink Models. Considering this, many conventional low level testing methods are adapted by various test engineers. This paper is to propose a method to undertake Low Level testing/ debugging in comparatively easier and faster way. As a first step, an attempt is made to simulate developed safety critical control blocks within a specified simulation time. For this, the blocks developed will be utilized to test in Simulink environment. What we propose here is Processor in loop test method using RTDX. The idea is to simulate model (requirement) in parallel with handwritten code (not a generated one) running on a specified target, subjected to same inputs (test cases). Comparing the results of model and target, fidelity can be assured. This paper suggests a development workflow starting with a model created in Simulink and proceeding through generating verified and profiled code for the processor.**

***Index Terms* — Coverage analysis, Embedded software, MATLAB, Safety critical system, Software, Software safety.**

I. Nomenclature

CCS	- Code Composer Studio
RTDX	- Real Time Data Exchange
PIL	- Processor in loop

I. Introduction

Modern electronic systems increasingly make use of embedded computer systems to add functionality, increase flexibility, controllability and performance. This can lead to new and different failure modes which cannot be addressed with traditional fault tolerance techniques [1]. This is especially significant in *Life/Safety critical system.*

Safety critical systems are those systems whose failure could result in loss of life, significant property damage, or damage to the environment. Safety-critical systems are increasingly computer-based and their development has traditionally been pioneered within the avionics and automotive industries.

The avionics industry has succeeded in producing standard methods for producing life-critical avionics software. The standard approach is to carefully code, inspect, document, test, verify and analyse the system [2].

In model based verification process, the models are manually transformed into code or with the use of autocode generators. The code is compiled and executed on the PC or on target boards and the outputs checked to ensure that they match the simulated model output [3]. Extensive testing is required for any safety critical system [4]. But failures still happen in flight controls.

This paper intends to change the thought from correctness-by-construction to construction-for-correctness. Any design related to safety critical application tends to be the safest thereby ensuring Correctness-by-construction. For this, once the system model is built, rigorous validation and verification procedures are to be followed. Instead an application can be assured to be the safest by following certain verification process throughout the construction.

In order to define tests capable of detecting errors and producing confidence in the software, test scenarios should be designed which are systematically based on the software requirements. However, requirement-based, i.e. logical, test scenarios are abstract and not executable [5]. This means that they cannot be used directly for test execution. Therefore, additional executable test scenarios are needed, which can stimulate the interface of the respective test object.

The test coverage is a function of software design assurance level. For a safety assured software system, test coverage of software requirements as a function of software design assurance level is nothing short of essentiality. Coverage refers to the extent to which a given verification activity has satisfied its objectives. Coverage is a measure, not a method or a test and is usually expressed as percentage of an activity that is accomplished. Appropriate coverage measures give verification activities a sense of the adequacy of the verification accomplished.

III. Processor In Loop

Processor in Loop (PIL) testing environment serve as a data exchange interface between the host simulation and the object code executing on the target. PIL simulation

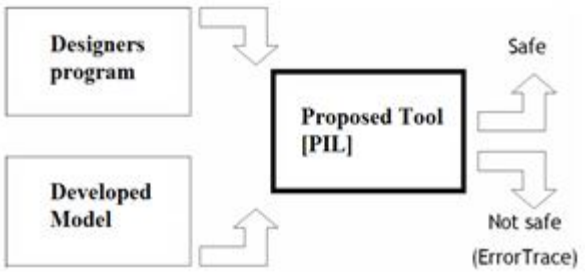

Fig. 1. Proposed Tool

DOI: 01.IJEPE.4.1.1079

capability functionally verifies models with the algorithm deployed on target platform without having to manually create an embedded test harness. The proposed tool using this capability is shown in fig.1.

PIL tests are close to real time but do not run in real time, as Simulink controls the execution of the PIL code on the target processor. The simulation halts during each sample period while data is transferred to the target processor. Once the object code completes executing on the processor, the data is transferred back to the host to resume host simulation. Through this approach, the functional difference between the model and compiled object code can be checked, to identify any potential deviations in the performance introduced by compilation process.

A. Model (Can be Requirement)

For an example, we developed a safety critical control block, (to be precise- Backlash) as per the requirements and is shown in fig.2. The functionality and behaviour of this block was thoroughly studied before getting into the testing.

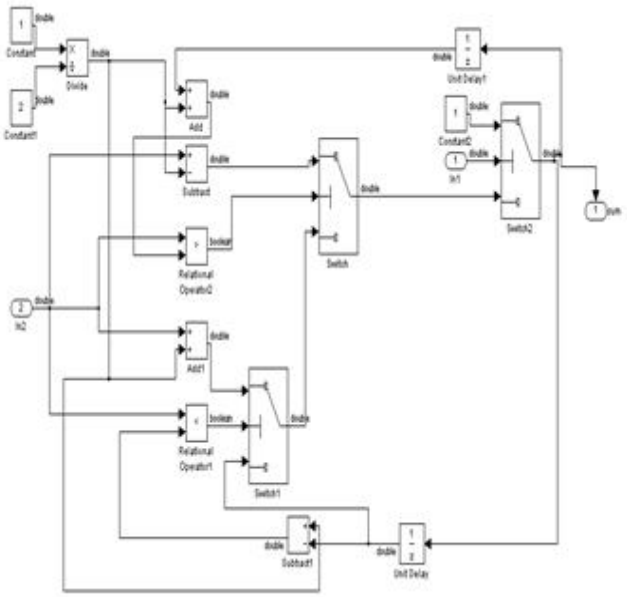

Fig. 2 Simulink Model for Backlash

Experience has shown that aircraft structures are generally affected by structural nonlinearities. The form of the nonlinearity encountered on actual aircraft structures is in general not very well known and is an area worthy of further research. In absence of more definite information, two relatively simple characteristic types of structural nonlinearities are: backlash and centrally symmetric hysteresis loop.

The amount by which the width of a tooth space exceeds the thickness of the engaging tooth on the pitch circles is its measure. As actually indicated by measuring devices, backlash may be determined variously in the transverse, normal, or axial planes, and wither in the direction of the pitch circles or on the line of action. Such measurements should be converted to corresponding values on transverse pitch circles for general comparison. The Backlash block implements a system in which a change in input causes an equal change in output. However, when the input changes direction, an initial change in input has no effect on the output. The amount of side-to-side play in the system is referred to as the *deadband*. The deadband is centered about the output.

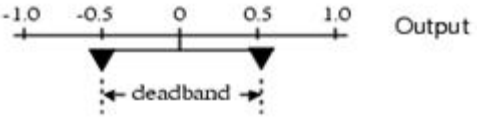

Fig. 3 Initial state of Backlash

The fig. 3 shows the block's initial state, with the default deadband width of 1 and initial output of 0 and fig. 4 shows the complete behaviour of backlash.

A system with play can be in one of three modes:

- *Disengaged* - In this mode, the input does not drive the output and the output remains constant.
- *Engaged in a positive direction* - In this mode, the input is increasing (has a positive slope) and the output is equal to the input *minus* half the deadband width.
- *Engaged in a negative direction* - In this mode ,the input is decreasing (has a negative slope) and the output is equal to the input *plus* half the deadband width.

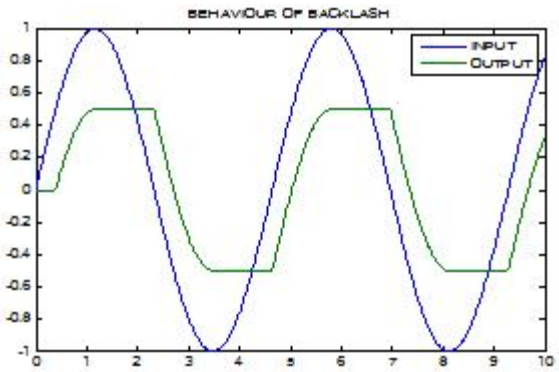

Fig. 4 Simulated behaviour of Backlash

B. C Code

For high quality software, it is important to choose the programming language and compiler to be used. There are likely to be a large number of different factors to take into account when making this choice, such as the past experience of the programmers; the re-use of existing modules; the compilers available, including the speed of execution of their generated code; and the portability of the programs [6]. One of the languages most commonly used in safety related applications and, therefore, the most often evaluated one in this respect is C [7]. Matlab share handles easily with CCS, thus developed code for the PIL environment is in C language.

A segment of developed C code indicating the functionality of the model is given below:

```
void bckfn(int init, double inp, state *a)
{
   double Out;
  Out = a->Out;
 if (init == 1)
```


DOI: 01.IJEPE.4.1.1079

```
    Out = ic;
 else
 {
if (inp > (Out + deadband/2))
     Out = inp - (deadband/2);

else if (inp < (Out - deadband/2))
     Out = inp + (deadband/2);
 }
a->Out = Out;

}
```

Now, the model is in Matlab/Simulink and C code in CCS. This necessitates establishing all the handles to start with PIL simulation. A function can be executed by calling it by means of the function handle.

Many such blocks where designed and each block is identified and determined to have separate functionality and its use is limited to the right system.

IV. Working With RTDX

RTDX offers continuous bi-directional data exchange in real time with minimal perturbation on the application. Basically its Texas Instrument's DSP analysis technology. RTDX provides continuous visibility into the way target applications operate in the real world. This helps us to get a realistic view of how the system will work.

RTDX establishes across the connection between MATLAB software and CCS, by which we can:

- Send and retrieve data from memory on the processor.
- Change the operating characteristics of the program.
- Make changes to algorithms as needed without stopping the program or setting breakpoints in the code (fig.5).

RTDX and Embedded IDE link CC provide a communication channel that enables to exchange data in real time or close to real time between Matlab and target simulator while target application is running (fig.6). System level test benches can be used to compare performance and results of code with the original reference model.

Embedded IDE link CC utilizes the in process communication and data transfer technology in CCS. As a result, application can read and write large data sets faster and quickly from and to the target. It supports PIL simulation with various processors using the same Matlab and Simulink test vectors for system design and verification.

V. Verification

Tests on PIL level are important because they can reveal faults that are caused by the target compiler or by the processor architecture. It is the last integration level which allows debugging during tests in a cheap and manageable way. The model and target implementation is randomly simulated to check any functional difference between them. Fig.7 shows the obtained results. Looking into the results obtained, it can be concluded that this approach of comparing

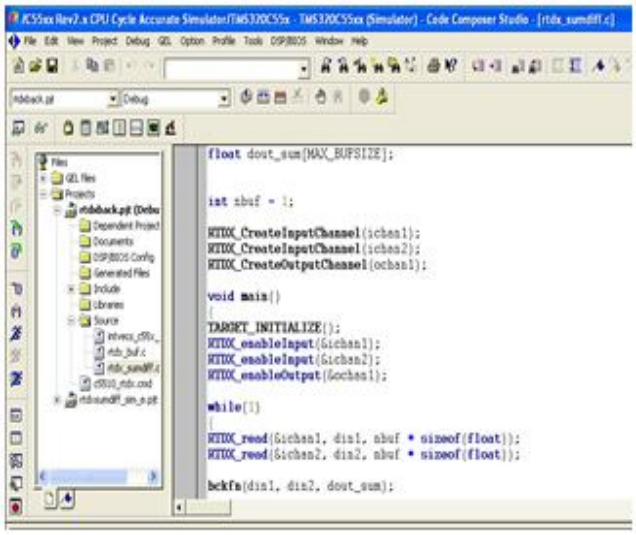

Fig. 5 RTDX initialisation in CCS

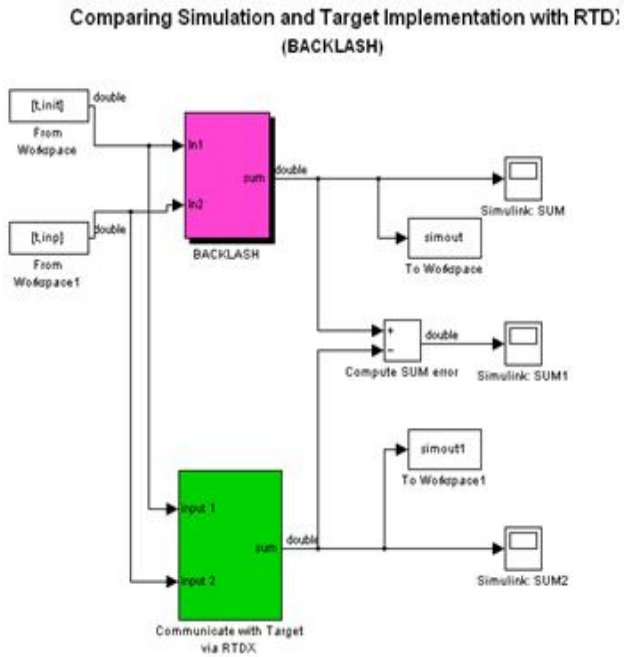

Fig. 6 Simulation with RTDX

the simulation and target implementation with RTDX can be easily adapted with proper understanding of the concept. Many improvements are to be made to our approach to standardize the same.

A. Coverage Analysis

Coverage is a form of testing that inspects the code directly and is a form of white box testing. Increasing the coverage increases the confidence in the design's correctness.

Code coverage describes the degree to which the source code of a program as been tested. One or more coverage criteria are used to measure how well the program is exercised by collection of test cases. The entire program was run with some test suites. Following is a bit of code developed for the system.

DOI: 01.IJEPE.4.1.1079

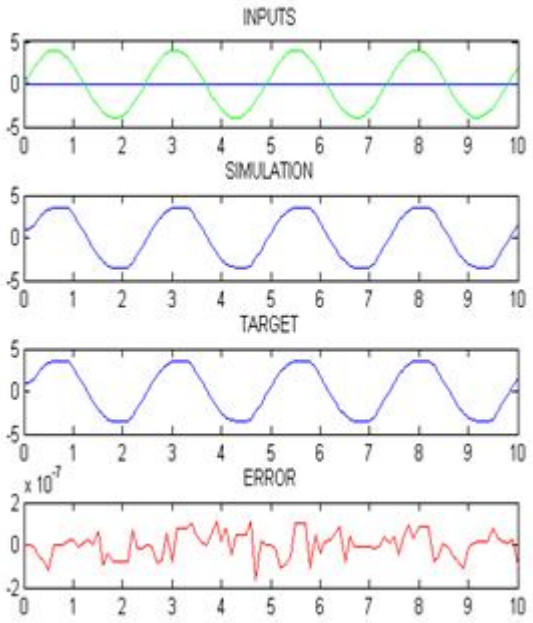

Fig. 7 Simulation Result

```
index = find(input>= (deadband));

if isempty(index)
.
.
index = find(input <= -(deadband/2));
if isempty(index)
    cov(2,1) = max((deadband/2 -
input(index)).^2);
    cov(2,2) = 0;
else
    cov(2,1) =  max((deadband/2 -
input(index)).^2);
.
.
.
end
```

Statement coverage is simplest form of coverage, which indicates that each and every statement is executed at least once. Functional coverage refers directly to the computation performed by the system. During simulation, each function call is exercised (here *max*).

Execution of blocks which act as a decision point is to be analyzed for decision coverage. To obtain full decision coverage, all the conditions in *if* statement is to be satisfied.

The complete code coverage analysis was performed by considering optimized test cases. The obtained simulation result is shown in fig. 8.

Conclusion

The basic control blocks of any safety critical system/ test specimen was developed and tested using random test cases. A C code was developed for the control blocks. Improvement and measurement of system quality involves the stages of test case specification, generation, execution

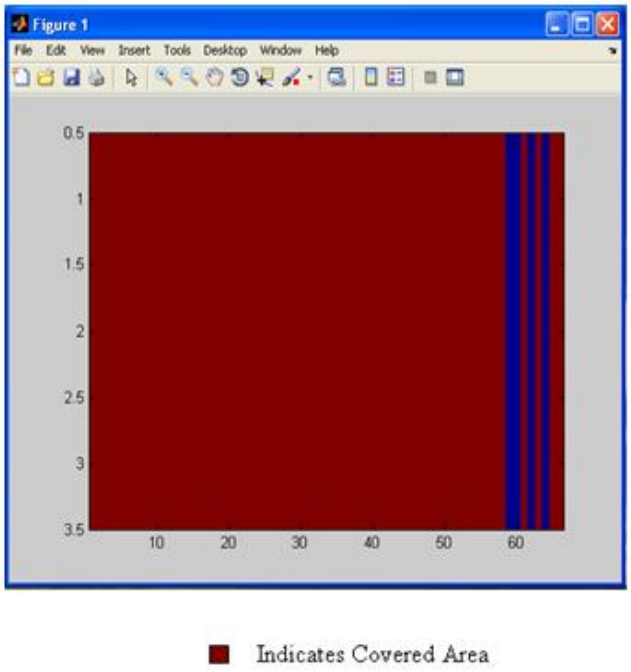

Fig. 8 Code coverage

and evaluation. A modeling and testing framework is proposed which successfully does a regular random testing to generate optimized test sets giving complete system coverage. In future, the PIL test method has to be extended for a processor board taking into consideration the real world inputs and noise in quantization.

References

[1] Embedded software design for safety critical systems, Knowledge transfer network electronics, Nov. 2009.

[2] Divyesh Divakar, Hariram Selvamurugan, Yogananda Jeppu, Nagaraj Murthy, Manjunath L, Shreesha Chokkadi, "Randomized Testing to Describe the Behaviour of Safety Critical Control Blocks", National Conference on Information Systems, 2011. "unpublished"

[3] Divyesh Divakar, K. Samatha, A. V. Veena Rani, Hariram Selvamurugan, Yogananda Jeppu, Nagaraj Murthy and Shreesha Chokkadi, "Optimization of Test case and Coverage Analysis for Model Based Design", 34TH National systems conference on System Solutions for Global Challenges: Energy, Environment and Security,10-12 December 2010. "unpublished"

[4] Johnson, C. W.; Holloway, C. M. "The Dangers of Failure Masking in Fault-Tolerant Software: Aspects of a Recent In-Flight Upset", Event NASA Center: Langley Research Center Publication Year 2007, Document ID: 20070034017.

[5] Mirko Conrad, Ines Fey, Sadegh Sadeghipour. "Systematic Model Based Testing of Embedded Automative Software", Science Direct Electronic Notes in Theoretical Computer Science 111 Publication Year 2005, page 13–26.

[6] C.J.Burgess, "Software quality issues when choosing a programming language", university of Bristol, England, 1995

[7] Wolfgang A. Halang, Janusz Zalewski, "Programming languages for use in safety-related applications", Annual Reviews in Control, 2003

DOI: 01.IJEPE.4.1.1079